Software Estimating Model Calibration

Dr. Randall Jensen
Software Engineering, Inc.

The last decade has brought increasing pressure on software model developers to include a calibration capability in their models that will reduce errors and improve software development estimates. An invalid underlying assumption is that software estimate errors are primarily due to weaknesses in the estimating technology (models). Data errors and the impact of estimator capability are never considered to be significant error sources. The intent of this article is to raise awareness of software estimate errors sources, and to objectively consider the real impacts of model calibration.

A common phrase used in many software presentations during the past few years is, "We need properly calibrated and validated models." This is an assumption that a perfect estimating model will eliminate the risk in scheduling and managing software projects. Unfortunately, the calibrated, validated model is only the tip of the estimating iceberg.

There are four error classes that exist in any software estimate: estimating technology (model), environment, project data, and the estimator. The estimating technology is centered on the estimating model or tool. No model is perfect; however, several widely used models are viable estimating tools.

The second error class involves the environment where the software is to be developed. The environment includes the development system such as tools and practices, operating system, physical facilities, organization structure, and management. This information, which is generally sketchy when the project acquisition estimates are made, is detailed and firm at the time of contract award.

Project data, the third error class, is a large error source in all estimates. Size is a fundamental estimating tool parameter and is usually specified in source lines of code. Effective size, which defines the project magnitude, is so difficult to establish for almost all estimates that alternate estimate forms such as function points and objects have become popular size predictors. No size predicting method is trivial, and none of the methods have proven accuracy advantages.

The last of the four error classes is the estimator. The estimator collects the estimate data, establishes the estimating model input parameters, performs the cost and schedule analysis, and produces the final estimate. A realistic estimate requires that the estimator be proficient with the estimating tool. Proficiency equates to training and experience. Training requires more than access to a User's Guide, and experience is not instantaneous.

The estimator and the project data are the largest error contributors. Technology is actually the least significant source of error. Any of the major models or tools in the hands of an experienced estimator can produce realistic software development estimates. The estimating tool is quite analogous to a scalpel in the hands of a surgeon.

Calibrated, Validated Tools

Those responsible for producing estimates typically make four simple assumptions. First, the estimator is not an accuracy issue. Second, the project data used to develop or calibrate the model is of high quality. Third, the environment is constant and not an issue ("Despite this cost variation, COCOMO does not include a factor for management quality, but instead provides estimates which assume the project will be well managed." [1]). Finally, the estimating technology (model) is assumed to be the major error source.

The phrase "properly calibrated and validated models" does not appear to be the elixir that eliminates the errors and risk in scheduling and managing software projects. Calibrated, validated models are still necessary for realistic software estimates, just as sharp scalpels are necessary in good surgery.

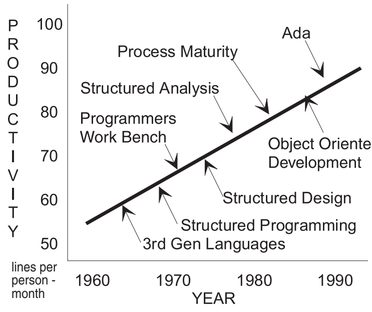
The estimating model must fit the data from which it is defined. This data must represent the estimating model application area. Developing a model from commercial data processing projects and applying the resulting model to spacecraft control should not be expected to produce acceptable cost and schedule predictions.

The major models are capable of estimating a wide range of software projects. These tools, in general, have been calibrated (and validated) with considerable project data over a long period of time. Internal environment adjustment factors account for variations between products and environments. For example, the number of environment adjustments in Sage and SEER-SEM is near 30. It is important to note these factors have also been validated. External model calibration changes the meanings of these factors and negates the model validation, as we will see later.

One argument against using traditional estimating tools is that traditional models were all developed in 1970s and 1980s. Fortunately, the software development industry has not changed significantly since then. True, we have much improved software development approaches and environments, but have these environments improved development productivity?

Defense industry software productivity, measured from the start of development through final qualification test, has grown almost linearly from 1960 through the present. A simplified (smoothed) productivity growth curve in Figure 1 shows this growth. The result, smoothed or not,

Figure 1: *Average Software Development Productivity Growth from 1960 to 1990*



shows a growth of software development productivity less than one source line per person-month per year during the entire 30-year period. For that period of time, each new technology has assured us the productivity problems of the past have been solved.

The traditional models, in the hands of experienced estimators still produce accurate and high-quality cost and schedule projections.

The Calibration Issues

The major calibration issue is what should be calibrated – the estimating model, the development organization, or the estimator. The industry trend leans toward the technology solution: the estimating model. The problem with the technology solution is calibration changes the model. This change is easily demonstrated with one of the simpler models, the Constructive Cost Model (COCOMO).

COCOMO vs. REVIC

The COCOMO software-estimating model was first published in 1981 [2]. The development effort *E,* defined by the embedded software version of the model, is given by Equation 1,

$$E = 2.8\prod_{i=1}^{16} f_i S_e^{1.2} \qquad (1)$$

where the impact of the product and the development environment is specified by the product of 16 effort adjustment factors

$$(\prod_{i=1}^{16} f_i)$$

and the effective development size . A variation of the COCOMO model, known as REVIC [3] was introduced a few years later. REVIC was developed from a collection of U.S. Air Force project data and defined development effort by the relationship shown in Equation 2,

$$E = 3.312\prod_{i=1}^{16} f_i S_e^{1.2} \qquad (2)$$

that differs from COCOMO (Equation 1) only in the proportionality constant. The REVIC equation is equivalent to

$$E = 2.8(k_1)\prod_{i=1}^{16} f_i S_e^{1.2}$$

where the *calibration factor k_1* =1.18. Note the REVIC definitions of the 16 environment adjustment factors (EAFs) are identical to the COCOMO definitions. REVIC was validated as a new estimating model using USAF project data. It is apparent that REVIC is a calibrated COCOMO model, and the calibration required that REVIC be revalidated before use to provide confidence in REVIC's cost and schedule estimates. The philosophical question, "Can any estimating model be changed and used without revalidation?" is a major concern. Many software-estimating tools contain calibration constants. Can these constants be changed from their default value without requiring model revalidation?

One serious model impact introduced by calibration is the redefinition of the model itself. We can illustrate the effect by factoring the analyst capability rating (ACAP) out of the Effort Adjustment Factor (EAF) product to obtain the equivalent effort relationship shown in Equation 3,

$$E = 2.8(k_1)(ACAP)\prod_{i=1}^{15} f_i S_e^{1.2} = \\ 2.8(1.0)(1.0)\prod_{i=1}^{15} f_i S_e^{1.2} \qquad (3)$$

For example, consider a software developer who uses COCOMO to produce realistic software estimates. The default calibration constant for COCOMO is 1.0. A manager notices the ACAP assumes the development organization to be average (*ACAP* = 1.0), which is unacceptable from a business standpoint.

The organization, according to the manager, should be considered to be in the 90th percentile category (*ACAP* = 0.71). Since the effort projections were realistic before the change to the ACAP, the calibration factor must be increased to 1.41 to rebalance the equation. The resulting effort equation is given by,

$$E = 2.8(1.41)(.71)\prod_{i=1}^{15} f_i S_e^{1.2} = 3.95\prod_{i=1}^{16} f_i S_e^{1.2}$$

Changing the proportionality constant from 2.8 to 3.95 is not the only change to the estimating model. The analyst capability definition was changed to allow average analysts to be rated in the upper 10 percentile as well. Is the model defined by the new effort equation equivalent to COCOMO? The models cannot be considered equivalent because the EAF definitions and the model definitions are different. Should the new equation define a new model that requires validation? Yes.

Calibration Database

The calibration (or validation) project database is also a major calibration issue. The database must be of sufficient size to effectively validate the new or modified estimating model. Inadequate data leads to inadequate validation. Yet, the calibration constant used in many models allows the validation database to be as small as a single project.

The data in the project database must also represent a common data definition; that is, defined with the same data rule set. For example, the task size definition must be consistent across all the development tasks in the database. It would be irrational to define part of the projects in terms of total size, part as modified size, and part as effective size.

The development environment for each task must be defined in the project database since no two tasks are developed under identical conditions. Some tasks may be developed in the Ada programming language, some may have severe timing constraints, and some may have a volatile requirements set. This information is necessary to adjust the environment factors in the selected model before a valid estimate can be produced or before the model can be validated (calibrated).

The U.S. Air Force Space and Missile Center, with assistance from the Space Systems Cost Analysis Group, developed a software project database that contains more than 2,800 contributed and unverified data records. The software project database is the largest and most widely used source of software project data. Some of the individual records provide fairly complete descriptions of each development task. This database has been the primary source of data for model calibration. Unfortunately, much of the data is incomplete, the definitions used by the individual data points are inconsistent, and the information necessary to extract effective size is not included. Thus, only a small portion of the software project database data is useful.

Estimator Impact

The most important variation in software estimates is the training, skill, and experience of the estimator. The impact of the estimator on an estimate is almost always ignored. Many estimators are not trained in estimating or in using their tools, which are assumed to be user-friendly (most are); the estimator need only to set values for the model parameters. The estimates are accepted without sanity checks to ascertain the estimate realism. It is easy to set model parameters, but sometimes quite difficult to set those parameters correctly. A slightly positive bias on all 16 COCOMO parameters can significantly change the resulting cost and schedule estimate.

Experience as an estimator or a software developer also has a great impact on the parameter values inserted into the estimate. Is it possible to understand development system volatility without some knowledge of the software development process? What is the impact of reuse or COTS on the effective size of the development? These questions demand experience and skill.

Decalogue Project

The Decalogue Project is the current stage of an ongoing software model calibration project that was started and first published in the early 1990s [4]. The project results have been published from 1995 through 1999 as Air Force Institute of Technology master's theses, and in other publications such as CrossTalk [5].

The project applies statistical methods to determine the accuracy of several software models. The current project results are not encouraging; however, the study results should be expected because of the underlying assumptions. The four simple estimate assumptions regarding estimators, data, environment, and estimating technology are all implicit in the Decalogue Project. The first indication that the Decalogue Project had problems was that *none* of the estimating models calibrated in the study had reasonable accuracy.

Before condemning the estimating model performance, the assumptions and experimental methods of the Decalogue Project must be understood. The most critical parameter in an estimate is the task size. The models use an effective size that is a weighted combination of new source code, original (existing) code, and modifications to the original code. The size data precision is very important to eliminate size inaccuracy from the estimate error.

The size data extracted from the software project database for the Decalogue Project was the total size data, not the effective size data that is used by the models. As an example of the estimate error introduced by using total size data, consider a system upgrade of 1,000 source lines in an existing 100,000-line software system. The development effort is more closely related to the 1,000-line upgrade than to the 100,000 existing lines. Development productivity is a simple test of the size data; that is, the ratio of size to development effort. More than half of the tested data points in the software project database yielded a productivity of more than 1,000 lines per person month. This productivity is an order of magnitude greater than that achieved in a typical project today.

The development environment was poorly specified or missing from the software project database. To compensate for the poor environment data, each model was set to a nominal environment description for the estimates. The nominal environment was essentially the model default.

Each software model was assigned to a graduate student for the analysis. Student quality is not an issue. However, the students had little, if any, specific model training or estimating experience.

The poor results achieved by the Decalogue Project demonstrate that poor calibration (validation) data, coupled with misused models and inexperienced estimators, lead to invalid estimates.

Conclusion

There are some fallacies related to the calibration of software estimating models. The first fallacy is that calibration data is generally accurate, consistent, and unbiased. Good validation data is carefully verified to ascertain its completeness, conformance to the database definitions, and accuracy. Size data in the calibration database must be complete enough to allow effective size to be extracted from the new, original, and modified size components.

Continued on page 18

Continued from page 15

Effective size, as used in the models, is necessary to predict cost and schedule. Organizational experience with the software also impacts effective size. Since organizations are not equal and unchanging, environment data is also necessary to perform estimates. Good validation data is not readily available.

The second fallacy is that estimates are independent of estimator training and experience. That is not likely. One cannot assume estimates produced by real people are ever above the significant biases introduced by lack of training, skill, or experience.

The third fallacy is that properly calibrated and validated tools eliminate major estimate errors; wrong again. Estimating tools are simply estimating tools. Proper validation reduces errors inherent in the tool itself. The model, or tool, is like a good scalpel. The quality of the estimate lies primarily in the quality of the user.

On the other hand, what disadvantages does calibration provide?

- Calibration invalidates the model. Model calibration requires revalidation before the model can be used with confidence.
- Calibration can destroy the meaning of the model's environment parameters. Unless that is the calibration goal, care must be taken to ensure the parameter definitions are intact.
- Calibration cloaks weakness in the estimator. Errors introduced by improperly using a model can be compensated for through the calibration process.
- Calibration makes it impossible for estimators using different versions (calibrations) of a model to compare results. This problem can get very severe when correlating estimates from the acquisition team and the contractor.

The bottom line is another phrase used in many software presentations during the past few years: "We need properly calibrated and validated models." Yes, we certainly need validated tools, but we also need trained, skilled, and experienced estimators.◆

References

1. Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, Inc., 1981, pg. 487.
2. Boehm, B. B., *Software Engineering Economics*, Prentice-Hall, Inc., 1981.
3. Kile, R.L., *REVIC Software Cost Estimating Model Users Manual*, Ver 9.0, February 9, 1991.
4. Ourada, G. L. and Ferens, D. V., Software Cost Estimating Models: A Calibration, Evaluation, and Comparison, *Cost Estimating and Analysis: Balancing Technology and Declining Budgets*, New York, Springer Verlag, 1992, pp. 83-101.
5. D. V. Ferens and Christensen, D. S., Does Calibration Improve Predictive Accuracy? CrossTalk, April 2000, pp. 14-17.

About the Author

Randall W. Jensen, Ph.D., is president of Software Engineering, Inc., and specializes in software project resource management. Dr. Jensen developed the model that underlies the Sage and the GAI SEER-SEM software cost and schedule estimating systems. He received the International Society of Parametric Analysts Freiman Award for Outstanding Contributions to Parametric Estimating in 1984. Dr. Jensen has published several textbooks, including *Software Engineering,* and numerous software and hardware analysis papers. He has bachelor's, master's and doctorate's degrees in electrical engineering from Utah State University.

Software Engineering, Inc.
660 North Highland Blvd.
Brigham City, UT 84302
Phone: (435) 734-2585
Fax: (435) 734-2586
E-mail: seisage@aol.com
www.seisage.com

> ***"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."***
>
> **—Maurice Wilkes**